

# Рекурентни уравнения

Тодор Дуков

## Защо са ни рекурентни уравнения?

Те се появяват по естествен път, когато искаме да анализираме сложността на рекурсивни алгоритми. Нека вземем за пример алгоритъма за двоично търсене:

```
1  int binary_search(int *arr, int left, int right, int val)
2  {
3      if (left > right)
4          return -1;
5
6      int mid = left + (right - left) / 2;
7
8      if (arr[mid] == val)
9          return mid;
10     else if (arr[mid] < val)
11         return binary_search(arr, mid + 1, right, val);
12     return binary_search(arr, left, mid - 1, val);
13 }
```

При подаден сортиран масив от числа `arr` с размер `n` и стойност `val`, функцията `binary_search(arr, 0, n - 1, val)` ще върне индекс на `arr`, в който се намира `val`, ако има такъв, иначе ще върне `-1`. Нека помислим каква е сложността на алгоритъма. Управляващите параметри на рекурсията са `left` и `right`. Всеки път разликата между двете намалява двойно (като накрая когато `left = right` тя ще стане отрицателна). Това означава, че в най-лошия случай сложността на алгоритъма може да се опише със следното рекурентно уравнение:

$$T(0) = 2 \text{ // заради ред 3 и 4}$$

$$T(n+1) = T(\lfloor \frac{n+1}{2} \rfloor) + 5 \text{ // заради проверките и рекурсивното извикване}$$

В този случай лесно се вижда асимптотиката на  $T(n)$ :

$$T(n) = T(\lfloor \frac{n}{2} \rfloor) + 5 = T(\lfloor \frac{n}{4} \rfloor) + 5 + 5 = T(\lfloor \frac{n}{8} \rfloor) + 5 + 5 + 5 = \dots = T(0) + \underbrace{5 + \dots + 5}_{\text{около } \log(n) \text{ пъти}} \asymp \log(n)$$

Така получаваме, че алгоритъмът има сложност  $O(\log(n))$ . Обаче в общият случай далеч не е толкова лесно да се намери асимптотичното поведение на дадено рекурентно уравнение. Целта ни ще бъде да развием по-богат апарат за асимптотичен анализ на рекурентните уравнения.

## Начини за намиране на асимптотиката на рекурентни уравнения

Начините се разделят на два типа:

1. със решаване на уравнението
2. без решаване на уравнението

И двата начина са ценни. Първият начин ни дава формула във явен вид, което може да ни е от полза. Понякога обаче формулата във явен вид не е “красива”, или изобщо не може да се намери такава. Тогава идва на помощ вторият начин. Той директно ни дава някаква “хубава” формула, без да трябва да намираме в явен вид решение на рекурентното уравнение. Проблема е обаче, че асимптотиката понякога е малко лъжлива – алгоритъм със сложност  $2^{2^{1000}}$  е асимптотично по-бавен от алгоритъм със сложност  $n$ , но практически вторият е по-бърз.

Ще разгледаме следните методи (повечето от които са разглеждани по дискретна математика):

- налучкване и доказване
- развиване (което преди малко показвахме)
- методът с характеристичното уравнение
- мастър-теоремата

Нека разгледаме един пример с налучкване:

$$T(0) = 3$$

$$T(n+1) = (n+1)T(n) - n$$

Започваме да разписваме:

$n$	$T(n)$	$n!$
0	3	1
1	3	1
2	5	2
3	13	6
4	49	24
5	241	120
6	1441	720

Вече лесно можем да покажем с индукция, че  $T(n) = 2(n!) + 1$ :

- $T(0) = 3 = 2 \cdot 1 + 1 = 2 \cdot 0! + 1$
- $T(n+1) = (n+1)T(n) - n \stackrel{(\text{ИП})}{=} (n+1)(2(n!) + 1) - n = (n+1)(2(n!)) + n + 1 - n = 2(n+1)! + 1$

Накрая получаваме, че  $T(n) \asymp n!$

Нека сега да видим как можем да използваме метода на характеристичното уравнение:

$$T(n) = 1 + \sum_{i=0}^{n-1} T(i) \quad // \text{ функцията е добре дефинирана и за } 0$$

Рекурентното уравнение, зададено в този вид, не може да се реши с този метод. За това ще трябва да направим преобразувания:

$$T(0) = 1$$

$$T(n+1) = 1 + \sum_{i=0}^n T(i) = 1 + T(n) + \sum_{i=0}^{n-1} T(i) = T(n) + \underbrace{\left(1 + \sum_{i=0}^{n-1} T(i)\right)}_{T(n)} = 2T(n) + 1 = \underbrace{2T(n)}_{\text{хомогенна част}}$$

Имаме само хомогенна част, от която получаваме характеристичното уравнение  $x - 2 = 0$  с единствен корен 2. Така:

$$T(n) = A \cdot 2^n \text{ за някоя константи } A$$

Вече няма нужда и да се намира константата – ясно е че  $T(n) \asymp 2^n$ . Като използваме метода на характеристичното уравнение, не е нужно да намираме накрая константите за да разберем каква е асимптотиката. Достатъчно е да вземем събираемостта, която расте най-много. В случая е ясно, че това е  $2^n$ .

Нека сега разгледаме и последният начин:

**Мастър-теорема.** Нека  $a \geq 1$ ,  $b > 1$  и  $f \in \mathcal{F}$ . Нека  $T(n) = aT(\frac{n}{b}) + f(n)$ , където  $\frac{n}{b}$  се интерпретира като  $\lfloor \frac{n}{b} \rfloor$  или  $\lceil \frac{n}{b} \rceil$ . Тогава:

1 сл. Ако  $f(n) \preceq n^{\log_b(a) - \varepsilon}$  за някое  $\varepsilon > 0$ , то тогава  $T(n) \asymp n^{\log_b(a)}$

2 сл. Ако  $f(n) \asymp n^{\log_b(a)}$ , то тогава  $T(n) \asymp n^{\log_b(a)} \log(n)$

3 сл. Ако са изпълнени следните условия:

1.  $f(n) \succeq n^{\log_b(a) + \varepsilon}$  за някое  $\varepsilon > 0$

2. съществува  $0 < c < 1$ , за което от някъде нататък  $a \cdot f(\frac{n}{b}) \leq c \cdot f(n)$ ,

то тогава  $T(n) \asymp f(n)$

Нека разгледаме рекурентното уравнение:

$$T(n) = 2T\left(\frac{n}{2}\right) + 1$$

Тук  $a = b = 2$ , и  $f(n) = 1$ . Също така  $\log_b(a) = 1$ , откъдето  $f(n) = 1 \preceq n^{\log_b(a) - \varepsilon}$ , за  $\varepsilon \in (0, 1)$ . Така по 1 сл. на мастър-теоремата получаваме, че  $T(n) \asymp n$ .

## Задачи

*Задача 1.* Да се намери асимптотиката на средната сложност по време на алгоритъма за бързо сортиране т.е. на рекурентното уравнение:

$$T(n) = \frac{1}{n} \left( \sum_{i=1}^{n-1} T(i) + T(n-i) \right) + n - 1$$

*Задача 2.* Да се намери асимптотиката на следните рекурентни уравнения:

$$\begin{aligned} T_1(n) &= 29T_1\left(\frac{n}{3}\right) + 2 \sum_{i=1}^n \frac{1}{i^2} & T_2(n) &= 29T_2\left(\frac{n}{3}\right) + 12n + \sqrt{n} & T_3(n) &= T_3(n-1) + \frac{n}{(n+1)(n-1)} \\ T_4(n) &= 29T_4\left(\frac{n}{3}\right) + \left(\sum_{i=1}^n \frac{1}{i}\right)^4 & T_5(n) &= 29T_5\left(\frac{n}{3}\right) + 2 \sum_{i=1}^n i^2 & T_6(n) &= 29T_6\left(\frac{n}{3}\right) + n^{\sqrt{n}} + (\sqrt{n})^n \\ T_7(n) &= T_7(\sqrt{n}) + n & T_8(n) &= 29T_8\left(\frac{n}{3}\right) + \binom{2n}{2} & T_9(n) &= 8T_9(n-1) - T_9(n-2) + 2n2^{2n} + 3n2^{3n} \end{aligned}$$

*Задача 3.* Да се намери сложността по време на следния алгоритъм:

```

1  int alg1(int n)
2  {
3      if (n < 2)
4          return n;
5
6      int acc = 0;
7
8      for (int i = 0; i < n; ++i)
9          {
10             acc += i;
11          }
12
13      return acc + alg1(n - 1) + alg1(n - 1) + alg1(n - 2);
14  }
```

Ще се промени ли нещо ако връщаме  $\text{acc} + 2 * \text{alg1}(n - 1) + \text{alg1}(n - 2)$ ?

*Задача 4.* Да се намери сложността по време на следния алгоритъм:

```

1  int alg2(int n)
2  {
3      if (n < 2)
4          return 2;
5
6      int t = 0;
7      t += alg2(n / 3);
8
9      for (int i = 2; i < n; i *= 2)
10         {
11             t++;
12         }
13
14      t *= alg2(n / 3);
15
16      return t;
17  }
```